

An Efficient Representation for Filtrations of Simplicial Complexes

Jean-Daniel Boissonnat*
 INRIA Sophia Antipolis - Méditerranée, France.
 Jean-Daniel.Boissonnat@inria.fr.

Karthik C. S.†
 Department of Computer Science and Applied Mathematics,
 Weizmann Institute of Science, Israel.
 karthik.srikanta@weizmann.ac.il.

Abstract

A filtration over a simplicial complex K is an ordering of the simplices of K such that all prefixes in the ordering are subcomplexes of K . Filtrations are at the core of Persistent Homology, a major tool in Topological Data Analysis. In order to represent the filtration of a simplicial complex, the entire filtration can be appended to any data structure that explicitly stores all the simplices of the complex such as the Hasse diagram or the recently introduced Simplex Tree [Algorithmica '14]. However, with the popularity of various computational methods that need to handle simplicial complexes, and with the rapidly increasing size of the complexes, the task of finding a compact data structure that can still support efficient queries is of great interest.

This direction has been recently pursued for the case of maintaining simplicial complexes. For instance, Boissonnat et al. [SoCG '15] considered storing the simplices that are maximal for the inclusion and Attali et al. [IJCGA '12] considered storing the simplices that block the expansion of the complex. Nevertheless, so far there has been no data structure that compactly stores the *filtration* of a simplicial complex, while also allowing the efficient implementation of basic operations on the complex.

In this paper, we propose a new data structure called the Critical Simplex Diagram (CSD) which is a variant of the Simplex Array List (SAL) [SoCG '15]. Our data structure allows to store in a compact way the filtration of a simplicial complex, and allows for the efficient implementation of a large range of basic operations. Moreover, we prove that our data structure is essentially optimal with respect to the requisite storage space. Next, we show that the CSD representation admits the following construction algorithms.

- A new *edge-deletion* algorithm for the fast construction of Flag complexes, which only depends on the number of critical simplices and the number of vertices.
- A new *matrix-parsing* algorithm to quickly construct relaxed Delaunay complexes, depending only on the number of witnesses and the dimension of the complex.

*This work was partially supported by the Advanced Grant of the European Research Council GUDHI (Geometric Understanding in Higher Dimensions).

†This work was partially supported by Irit Dinur's ERC-StG grant number 239985.

1 Introduction

Persistent homology is a method for computing the topological features of a space at different spatial resolutions [EH10]. More persistent features are detected over a wide range of length and are deemed more likely to represent true features of the underlying space, rather than artifacts of sampling, noise, or particular choice of parameters. To find the persistent homology of a space [BDM15, BM14a], the space is represented as a sequence of simplicial complexes called a filtration. The most popular filtrations are nested sequences of increasing simplicial complexes but more advanced types of filtrations have been studied where consecutive complexes are mapped using more general simplicial maps [DFW14]. Persistent homology found applications in many areas ranging from image analysis [CidSZ08, PC14], to cancer research [ABD⁺12], virology [CCR13], and sensor networks [dSG07].

Thus, a central question in Computational Topology and Topological Data Analysis is to represent simplicial complexes and filtrations efficiently. The most common representation of simplicial complexes uses the Hasse diagram of the complex that has one node per simplex and an edge between any pair of incident simplices whose dimensions differ by one. A more compact data structure, called Simplex Tree (ST), was proposed recently by Boissonnat and Maria [BM14b]. The nodes of both the Hasse diagram and ST are in bijection with the simplices (of all dimensions) of the simplicial complex. In this way, they explicitly store all the simplices of the complex and it is easy to attach information to each simplex (such as a filtration value). In particular, they allow to store in a natural way the filtration of complexes which are at the core of Persistent Homology and Topological Data Analysis.

However, such data structures are redundant and typically very big, and they are not sensitive to the underlying structure of the complexes. This motivated the design of more compact data structures that represent only a sufficient subset of the simplices. A first idea is to store the 1-skeleton of the complex together with a set of blockers that prevent the expansion of the complex [ALS12]. A dual idea is to store only the simplices that are maximal for the inclusion. Following this last idea, Boissonnat et al. [BKT16] introduced a new data structure, called Simplex Array List, which was the first data structure whose size and query time were sensitive to the geometry of the simplicial complex. SAL was shown to outperform ST for a large class of simplicial complexes.

Although very efficient, SAL, as well as data structures that do not explicitly store all the simplices of a complex, makes the representation of filtrations problematic, and in the case of SAL, impossible. In this paper, we introduce a new data structure called Critical Simplex Diagram (CSD) which has some similarity with SAL. CSD only stores the critical simplices, i.e., those simplices all of whose cofaces have a higher filtration value, and in this paper, we overcome the problems arising due to the implicit representation of simplicial complexes, by showing that the basic operations on simplicial complexes can be performed efficiently using CSD. In short, CSD compromises on the membership query (which is slightly worse than that for ST) in order to save storage and to perform insertion and removal efficiently.

1.1 Our Contribution

At a high level, our main contribution through this paper is in developing a new perspective for the design of data structures representing simplicial complexes associated with a filtration. Previous data structures such as Hasse diagram and Simplex Tree interpreted a simplicial com-

plex as a set of strings defined over the label set of its vertices and the filtration values as keys associated with each string. When a simplicial complex is perceived this way, a Trie is indeed a natural data structure to represent the complex. However, this way of representing simplicial complexes doesn't make use of the fact that simplicial complexes are not arbitrary sets of strings but are constrained by a lot of combinatorial structure. In particular, simplicial complexes are closed under subsets and also (standard) filtrations are monotone functions. We exploit these constraints/structure by viewing a filtered simplicial complex with filtration range of size t as a monotone function from $\{0, 1\}^{|V|}$ to $\{0, 1, \dots, t\}$, where V is the vertex set. We note that if a simplex is mapped to t then, the simplex is understood to be not in the complex and if not, the mapping is taken to correspond to the filtration value of the simplex. In light of this viewpoint, we propose a data structure (CSD) which stores only the critical elements in the domain, i.e. those elements all of whose supersets (cofaces in the complex) are mapped to a strictly larger value. As a result, we are not only able to store data regarding a simplicial complex more efficiently but also explicitly utilize geometric regularity in the complex which would have been otherwise obscured. More concretely, we have the following result.

Theorem 1. *Let K be a d -dimensional simplicial complex. Let κ be the number of critical simplices in the complex. The data structure CSD representing K admits the following properties:*

- *The size of CSD is at most κd .*
- *The cost of basic operations (such as membership, insertion, removal, elementary collapse, etc.) through the CSD representation is $\tilde{O}((\kappa \cdot d)^2)$.*

The proof of the above two items follows from the discussions in Section 3.3 and Section 3.4 respectively. We would like to point out here that while the cost of static operations such as membership is only $\tilde{O}(d)$ for the Simplex Tree, to perform any dynamic operation such as insertion or removal, the Simplex Tree requires $\exp(d)$ time. Moreover, as shown in Section 3.4, the cost of *most* basic operation using CSD is linear in κ .

As a direct consequence of representing a simplicial complex only through the critical simplices, we note that the construction of any simplicial complex with filtration, will be very efficient through CSD, simply because we have to build a smaller data structure as compared to the existing data structures. More specifically, we propose a new *edge-deletion* algorithm for the construction of flag complexes on n vertices with κ critical simplices in time $\mathcal{O}(\kappa n^{2.38})$. Additionally, we provide a *matrix-parsing* algorithm for building d -dimensional relaxed Delaunay complexes over the witness set W in $\mathcal{O}(|W|d^2 \log |W|)$ time. In each of these cases, we show that the construction is more efficient when using CSD rather than ST, primarily because CSD is a compact representation.

1.2 Organization of the paper

In Section 2, we introduce definitions which will be used throughout the paper and we provide new lower bounds on the space of data structures needed to store simplicial complexes. In Section 3, we introduce the new data structure CSD to represent simplicial complexes and their filtration. Further, we describe how the data structure can perform basic operations on the complex and also discuss about its efficiency. In Section 4, we give a new algorithm for the construction of Flag complexes. In Section 5, we give a new algorithm for the construction of relaxed Delaunay complexes. Finally, in Section 6, we conclude by highlighting some open directions for future research.

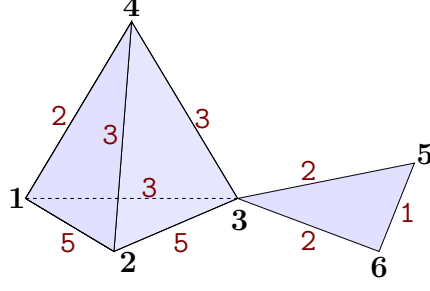


Figure 1: A simplicial complex with two maximal simplices : tetrahedron [1234] and triangle [356]. The filtration value of all vertices is 0. Filtration values of edges are shown in the figure. The filtration value of a higher dimensional simplex is the largest filtration value of its edges.

2 Preliminaries

A simplicial complex K is defined over a (finite) vertex set V whose elements are called the vertices of K and is a set of non-empty subsets of V that is required to satisfy the following two conditions:

1. $p \in V \Rightarrow \{p\} \in K$
2. $\sigma \in K, \tau \subseteq \sigma \Rightarrow \tau \in K$

Each element $\sigma \in K$ is called a simplex or a face of K and, if $\sigma \in K$ has precisely $s + 1$ elements ($s \geq -1$), σ is called an s -simplex and the dimension of σ , denoted by d_σ is s . The dimension of the simplicial complex K is the largest d such that it contains a d -simplex.

A *face* of a simplex $\sigma = \{p_0, \dots, p_s\}$ is a simplex whose vertices form a subset of $\{p_0, \dots, p_s\}$. A proper face is a face different from σ and the facets of σ are its proper faces of maximal dimension. A simplex $\tau \in K$ admitting σ as a face is called a *coface* of σ . A maximal simplex of a simplicial complex is a simplex which has no cofaces. A simplicial complex is pure, if all its maximal simplices are of the same dimension. Also, a free pair is defined as a pair of simplices (τ, σ) in K where τ is the only coface of σ .

In this paper, the class of simplicial complexes of n vertices in d dimensions with k maximal simplices out of the m simplices in the complex is denoted by $\mathcal{K}(n, d, k, m)$, and K denotes a simplicial complex in $\mathcal{K}(n, d, k, m)$.

In Figure 1 we see a three dimensional simplicial complex on the vertex set $\{1, 2, 3, 4, 5, 6\}$. This complex has two maximal simplices: the tetrahedron [1234] and the triangle [356]. We use this complex as an example through out the paper.

We adopt the following notation through out the paper: $[t] := \{1, \dots, t\}$ and $\llbracket t \rrbracket = \{0, 1, \dots, t\}$. A *filtration* over a simplicial complex K is an ordering of the simplices of K such that all prefixes in the ordering are subcomplexes of K . More concretely, a filtration of a complex is a function $f : K \rightarrow \mathbb{R}$ satisfying $f(\tau) \leq f(\sigma)$ whenever $\tau \subseteq \sigma$ [EH10]. Moreover, we will assume that the filtration values range over $\llbracket t \rrbracket$. We say that a simplex $\sigma \in K$ is a ‘**critical simplex**’ if for all cofaces τ of σ we have $f(\sigma) < f(\tau)$. For example, the critical simplices in the example described in Figure 1 are all the vertices, the edges [56], [14], and [24], the triangles [356] and [134], and the tetrahedron [1234].

2.1 Lower Bounds

Boissonnat et al. proved the following lower bound on the space needed to represent simplicial complexes [BKT16].

Theorem 2. [BKT16] *Consider the class of all d -dimensional simplicial complexes with n vertices containing k maximal simplices, where $d \geq 2$ and $k \geq n + 1$, and consider any data structure that can represent the simplicial complexes of this class. Such a data structure requires $\log \binom{n/2}{k-n}$ bits to be stored. For any constant $\varepsilon \in (0, 1)$ and for $\frac{2}{\varepsilon}n \leq k \leq n^{(1-\varepsilon)d}$ and $d \leq n^{\varepsilon/3}$, the bound becomes $\Omega(kd \log n)$.*

We prove now a lower bound on the representation of filtrations of simplicial complexes.

Lemma 3. *Let $\beta = \lfloor \frac{t+1}{d+1} \rfloor$ be greater than 1. For any simplicial complex K of dimension d containing m simplices, the number of distinct filtrations $f : K \rightarrow \llbracket t \rrbracket$ is at least β^m . If $\beta > (d+1)^\delta$ for some constant $\delta > 0$ then, any data structure that can represent filtrations of the class of all d -dimensional simplicial complexes containing m simplices requires $\Omega(m \log t)$ bits to be stored.*

Proof. Let us fix a simplicial complex K of dimension d containing m simplices. We will now build functions $f_i : K \rightarrow \llbracket t \rrbracket$. For every $i \in \llbracket \beta^m - 1 \rrbracket$, let $b(i)$ be the representation of i as a m digit number in base β and let $b(i)_j$ be the j^{th} digit of $b(i)$. Let g be a bijection from K to $[m]$. We define $f_i(\sigma) = d_\sigma \cdot \beta + b(i)_{g(\sigma)} + 1$. We note that all the f_i s are distinct functions as for any two distinct numbers i and j in $\llbracket \beta^m - 1 \rrbracket$, we have that $b(i) \neq b(j)$. Finally, we note that each of the f_i s is a filtration of K . This is because, for any two simplices $\tau, \sigma \in K$, such that $\tau \subset \sigma$ (i.e., $d_\tau < d_\sigma$), and any $i \in \llbracket \beta^m - 1 \rrbracket$, we have that $f_i(\tau) \leq (d_\tau + 1) \cdot \beta < d_\sigma \cdot \beta + 1 \leq f_i(\sigma)$.

It follows that there are at least β^m distinct filtrations of K . By the pigeonhole principle, we have that any data structure that can represent filtrations of K requires $\log(\beta^m)$ bits. It follows that if $\beta > (d+1)^\delta$ for some constant $\delta > 0$ then $\beta > \frac{(t+1)^\delta}{(\beta-1)^\delta}$. Thus, any data structure that can represent filtrations of K requires at least $\frac{\delta}{1+\delta} \cdot m \log(t+1) = \Omega(m \log t)$ bits to be stored. \square

Even if $\lfloor \frac{t}{d} \rfloor \leq 1$, we can show that any data structure that can represent d -dimensional simplicial complexes containing m simplices with filtration range $\llbracket t \rrbracket$ requires $\Omega(\frac{m\sqrt{t}}{d} \log t)$ bits. This can be shown by modifying the above proof as follows. Let S_j be the set of all simplices of dimension $j-1$. We identify a subset D of $[d]$ of size \sqrt{t} , such that $\sum_{j \in D} |S_j|$ is at least $\frac{m\sqrt{t}}{d}$. Therefore, for every set S_j , $j \in D$, we can associate \sqrt{t} distinct filtration values, which leads to the lower bound.

The lower bounds in Lemma 3 is not sensitive to the number of critical simplices, and intuitively, any lower bound on the size of data structures storing complexes with filtrations needs to capture the number of critical simplices as a parameter. We adapt the proof of Theorem 2 and combine it with the ideas from the proof of Lemma 3, to obtain the following lower bound.

Theorem 4. *Consider the class of all simplicial complexes on n vertices of dimension d , associated with a filtration over the range of $\llbracket t \rrbracket$, such that the number of critical simplices is κ , where $d \geq 2$ and $\kappa \geq n+1$, and consider any data structure that can represent the simplicial complexes*

of this class. Such a data structure requires $\log \left(\binom{\binom{n/2}{d+1}}{\kappa-n} t^{\kappa-n} \right)$ bits to be stored. For any constant $\varepsilon \in (0, 1)$ and for $\frac{2}{\varepsilon}n \leq \kappa \leq n^{(1-\varepsilon)d}$ and $d \leq n^{\varepsilon/3}$, the bound becomes $\Omega(\kappa(d \log n + \log t))$.

Proof. The proof of the first statement is by contradiction. Let us define $h = \kappa - n \geq 1$ and suppose that there exists a data structure that can be stored using only $s < \log \alpha \stackrel{\text{def}}{=} \log \left(\binom{\binom{n/2}{d+1}}{\kappa-n} t^h \right)$ bits. We will construct α simplicial complexes (associated with a filtration), all with the same set P of n vertices, the same dimension d , with exactly κ maximal simplices, and with a filtration over the range of $[t]$. By the pigeon hole principle, two different simplicial complexes*, say K and K' , are encoded by the same word. So any algorithm will give the same answer for K and K' . But, by the construction of these complexes, there is either a simplex which is in K and not in K' or there is a simplex whose filtration value in K is different from the simplex's filtration value in K' . This leads to a contradiction.

The simplicial complexes and their associated filtration are constructed as follows. Let $P' \subset P$ be a subset of cardinality $n/2$, and consider the set of all possible simplicial complexes of dimension d with vertices in P' that contain h critical simplices. We further assume that all critical simplices have dimension d exactly. These complexes are $\beta = \binom{\binom{n/2}{d+1}}{h}$ in number, since the total number of maximal d dimensional simplices is $\binom{n/2}{d+1}$ and we choose h of them. Let us call them $\Gamma_1, \dots, \Gamma_\beta$. We now extend each Γ_i so as to obtain a simplicial complex whose vertex set is P and has exactly κ critical simplices. The critical simplices will consist of the h maximal simplices of dimension d already constructed (whose filtration value is set to one of the values in $[t]$) plus a number of maximal simplices of dimension 1 (whose filtration value is set to 0). The set of vertices of Γ_i , $\text{vert}(\Gamma_i)$, may be a strict subset of P' . Let its cardinality be $\frac{n}{2} - r_i$ and observe that $0 \leq r_i < \frac{n}{2}$. Consider now the complete graph on the $\frac{n}{2} + r_i$ vertices of $P \setminus \text{vert}(\Gamma_i)$. Any spanning tree of this graph gives $\frac{n}{2} + r_i - 1$ edges and we arbitrarily choose $\frac{n}{2} - r_i + 1$ edges from the remaining edges of the graph to obtain n distinct edges spanning over the vertices of $P \setminus \text{vert}(\Gamma_i)$. We have thus constructed a 1-dimensional simplicial complex K_i on the $\frac{n}{2} + r_i$ vertices of $P \setminus \text{vert}(\Gamma_i)$ with exactly n maximal simplices. Finally, we define the complex $\Lambda_i = \Gamma_i \cup K_i$ that has P as its vertex set, dimension d , and κ maximal simplices which are also the critical simplices. The filtration value of any simplex which is not maximal is defined to be the minimum of the filtration values of its cofaces in the complex. The set of Λ_i , $i = 1, \dots, \beta$, where for each complex we associate t^h different filtrations is the set of simplicial complexes (associated with a filtration) that we were looking for.

The second statement in the theorem is proved through the following computation:

$$\begin{aligned} \log \left(\binom{\binom{n/2}{d+1}}{\kappa-n} t^{\kappa-n} \right) &\geq \log \left(\frac{n^{(d+1)(\kappa-n)}}{2^{(d+1)(\kappa-n)} (d+1)^{(d+1)(\kappa-n)} (\kappa-n)^{(\kappa-n)}} \right) + (\kappa-n) \log t \\ &= (d+1)(\kappa-n) \log n - (d+1)(\kappa-n) - (d+1)(\kappa-n) \log(d+1) \\ &\quad - (\kappa-n) \log(\kappa-n) + (\kappa-n) \log t \\ &> (d+1)(\kappa-n) \log n - 3(d+1)(\kappa-n) - (d+1)(\kappa-n) \log d \\ &\quad - (\kappa-n) \log \kappa + (\kappa-n) \log t \end{aligned}$$

*i.e., two complexes which either differ on the simplices contained on the complex or on the filtration value of a simplex contained in both complexes.

$$\begin{aligned}
&\geq (d+1)(\kappa-n)(\log n - 3 - \log d) - (\kappa-n)(1-\varepsilon)d \log n \\
&\quad + (\kappa-n) \log t \\
&\geq d\varepsilon(\kappa-n) \log n + (\kappa-n) \log n - (d+1)(\kappa-n) \left(3 + \frac{\varepsilon}{3} \log n\right) \\
&\quad + (\kappa-n) \log t \\
&\geq \frac{2\varepsilon}{3} \left(1 - \frac{\varepsilon}{2}\right) \kappa d \log n + \left(1 - \frac{\varepsilon}{2}\right) \kappa \log t + \left(1 - \frac{\varepsilon}{2}\right) \kappa \log n \\
&\quad - 3d \left(1 - \frac{\varepsilon}{2}\right) \kappa - \left(1 - \frac{\varepsilon}{2}\right) \kappa \left(3 + \frac{\varepsilon}{3} \log n\right) \\
&= \Omega(\kappa(d \log n + \log t))
\end{aligned}$$

We note that in the above computation, the first inequality is obtained by applying the following bound on binomial coefficients: $\binom{n}{d} \geq \left(\frac{n}{d}\right)^d$. \square

2.2 Simplex Tree

Let $K \in \mathcal{K}(n, d, k, m)$ be a simplicial complex whose vertices are labeled from 1 to n and ordered accordingly. We can thus associate to each simplex of K a word on the alphabet set $[n]$. Specifically, a j -simplex of K is uniquely represented as the word of length $j+1$ consisting of the ordered set of the labels of its $j+1$ vertices. Formally, let $\sigma = \{v_{\ell_0}, \dots, v_{\ell_j}\}$ be a simplex, where v_{ℓ_i} are vertices of K and $\ell_i \in [n]$ and $\ell_0 < \dots < \ell_j$. σ is represented by the word $[\sigma] = [\ell_0, \dots, \ell_j]$. The simplicial complex K can be defined as a collection of words on an alphabet of size n . To compactly represent the set of simplices of K , the corresponding words are stored in a tree and this data structure is called the Simplex Tree of K and denoted by $\text{ST}(K)$ or simply ST when there is no ambiguity. It may be seen as a trie on the words representing the simplices of the complex. The depth of the root is 0 and the depth of a node is equal to the dimension of the simplex it represents plus one.

We give a constructive definition of ST . Starting from an empty tree, insert the words representing the simplices of the complex in the following manner. When inserting the word $[\sigma] = [\ell_0, \dots, \ell_j]$ start from the root, and follow the path containing successively all labels ℓ_0, \dots, ℓ_i , where $[\ell_0, \dots, \ell_i]$ denotes the longest prefix of $[\sigma]$ already stored in the ST . Next, append to the node representing $[\ell_0, \dots, \ell_i]$ a path consisting of the nodes storing labels $\ell_{i+1}, \dots, \ell_j$. The filtration value of σ denoted by $f(\sigma)$ is stored inside the node containing the label ℓ_j , in the above path. In Figure 2, we give ST for the simplicial complex shown in Figure 1.

If K consists of m simplices (including the empty face) then, the associated ST contains exactly m nodes. Thus, we need $\Theta(m(\log n + \log t))$ space/bits to represent the nodes in ST (since each node stores a vertex which needs $\Theta(\log n)$ bits to be represented and also stores the filtration value of the simplex that the node corresponds to, which needs $\Theta(\log t)$ bits to be represented). We remark here that we don't consider the space needed to maintain the edges of ST , as it is part of the data structure implementation of ST . We can compare the upper bound obtained to the lower bound of Lemma 3. In particular, ST matches the lower bound, when t is not small.

Now, we will briefly recapitulate the cost of doing some basic operations through ST on a simplicial complex. To check if a simplex σ is in the complex, is equivalent to checking the existence of the corresponding path starting from the root in ST . This can be done very efficiently in time $\mathcal{O}(d_\sigma \log n)$ and therefore all operations which primarily determine on the

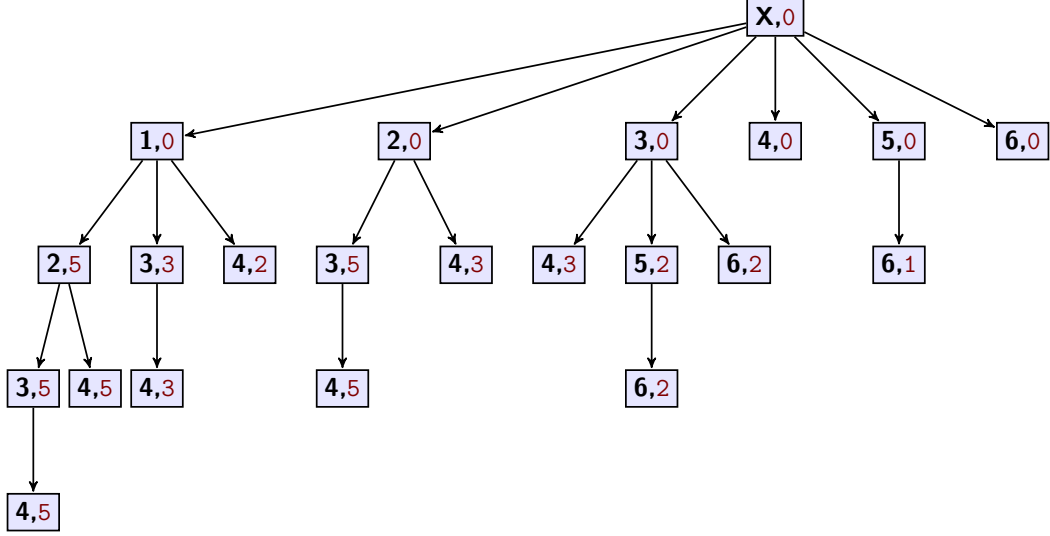


Figure 2: Simplex Tree of the simplicial complex in Figure 1. In each node, the label of the vertex is indicated in black font and the filtration value stored by the simplex is in brown font.

membership query can be efficiently performed using ST. One such example, is querying the filtration value of a simplex. However, due to its explicit representation, insertion is a costly operation on ST (exponential in the dimension of the simplex to be inserted). Similarly, removal is also a costly operation on ST, since there is no efficient way to locate and remove all cofaces of a simplex. Consequently, topology preserving operations such as elementary collapse and edge contraction are also expensive for ST. These operation costs are summarized later in Table 1. In the next section, we will introduce a new data structure which does a better job of balancing between static queries (e.g. membership) and dynamic queries (e.g. insertion and removal).

3 Critical Simplex Diagram

In this section, we introduce the *Critical Simplex Diagram*, $\text{CSD}(K)$, which is an evolved version of SAL [BKT16]. CSD is a collection of n arrays that correspond to the n vertices of K . The elements of an array, referred to as *nodes* in the rest of the paper, correspond to copies of the vertex of K associated to the array. Additionally, CSD has edges that join nodes of different arrays. Each connected component of edges in CSD represents a simplex of K . Not all simplices of K are represented but only those simplices all of whose cofaces have a higher filtration value. We describe some notations used throughout the section below.

3.1 Notations

Let S_h be the set of simplices in the complex whose filtration value is h . Let M_h be the maximal subset of S_h containing all the critical simplices of S_h . For instance, in the complex of Figure 1, we have $M_0 = \{1, 2, 3, 4, 5, 6\}$, $M_1 = \{[56]\}$, $M_2 = \{[14], [356]\}$, $M_3 = \{[134], [24]\}$, $M_4 = \emptyset$, and $M_5 = \{[1234]\}$. Moreover, we note that M_0, M_1, \dots, M_t are all disjoint and denote by M the union of all M_h .

We denote by $\Psi_{\max}(i)$ the subset of simplices of M_h that contain vertex i for all possible

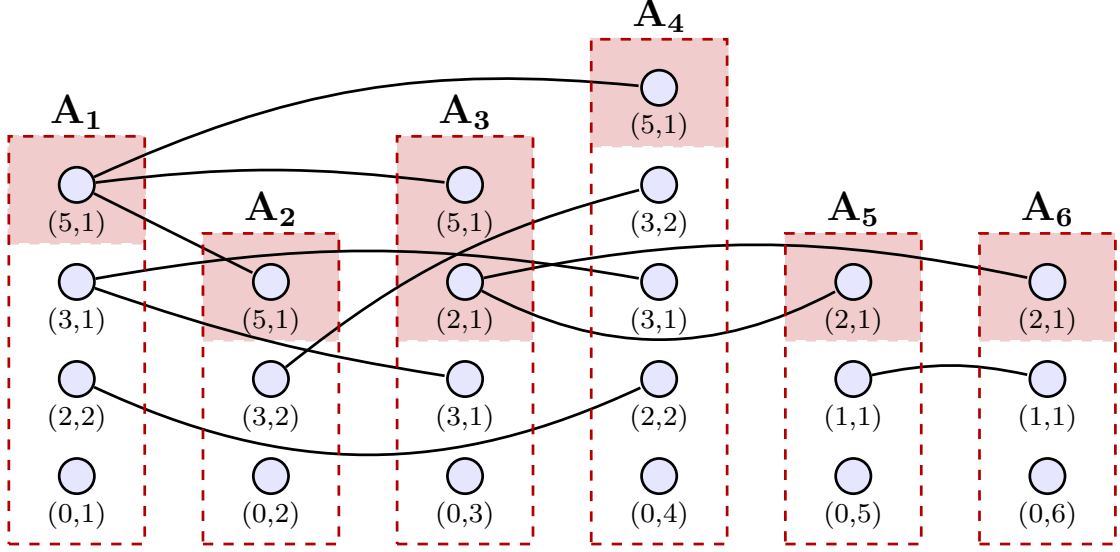


Figure 3: Critical Simplex Diagram for the complex in Figure 1. The shaded region in each A_i corresponds to A_i^* .

$h \in \llbracket t \rrbracket$. In other words,

$$\Psi_{\max}(i) = \{\sigma \in K \mid i \in \sigma, f(\sigma) < f(\tau), \forall \tau \in K, \sigma \subset \tau\}.$$

Let $\Psi = \max_{i \in [n]} |\Psi_{\max}(i)|$.

We denote by Γ_j the largest number of maximal simplices of K that a given j -simplex of K may be contained in. We note the following bounds:

$$k \geq \Gamma_0 \geq \Gamma_1 \geq \dots \geq \Gamma_d = 1,$$

$$\Gamma_0 \leq \Psi \leq m.$$

Moreover, when $t = 0$, we have $\Psi = \Gamma_0$. We describe the construction of CSD below.

3.2 Construction

We initially have n empty arrays A_1, \dots, A_n . The vertices of K are associated to the arrays. Each array contains a set of nodes that are copies of the vertex associated to the array and are labelled by an ordered pair of integers (to be defined below). Nodes belonging to distinct arrays are joined by edges leading to a graph structure. The connected components of that graph represent the simplices in M_0, M_1, \dots, M_5 . All the nodes of such a simplex are labelled by a pair of integers. The first integer refers to the filtration value of the simplex and the second integer refers to a number used to index simplices that have the same filtration value. For instance, in Figure 3 we have the CSD representation of the simplicial complex of Figure 1, and the triangle $[134]$ in M_3 is represented by 3 nodes, each with label $(3, 1)$, that are connected by edges. Below we provide a more detailed treatment of the construction of CSD.

Given M_h for every $h \in \llbracket t \rrbracket$, we build the CSD by inserting the simplices in M_h in decreasing ordering of h . For every simplex $\sigma = v_{\ell_0} \dots v_{\ell_j}$ in M_h , we associate a unique key generated using a hash function \mathcal{H} , $\mathcal{H}(\sigma) \in \llbracket M_h \rrbracket$, and insert the nodes with label $(h, \mathcal{H}(\sigma))$ in to the arrays $A_{\ell_0}, \dots, A_{\ell_j}$. For every $j' \in [j]$, we introduce an edge between $(h, \mathcal{H}(\sigma))$ in A_{ℓ_0} and $(h, \mathcal{H}(\sigma))$ in

A_{ℓ_j} . In other words, σ , a critical simplex is being represented in CSD by a connected component in the graph thus defined. Each connected component is a star graph on $d_\sigma + 1$ nodes where v_{ℓ_0} is the center of the star. Furthermore, each node in CSD corresponds to a vertex in exactly one simplex.

We denote by A_i^* the set of all nodes in A_i of star graphs that correspond to maximal simplices in K (the region with these nodes are shaded in Figure 3). Inside each A_i , we first sort nodes based on whether they are in A_i^* or not. Further, inside A_i^* and inside $A_i \setminus A_i^*$, we sort the nodes according to the lexicographic order of their labels. We note that A_i^* is a contiguous subarray of A_i , i.e., all consecutive elements in A_i^* are also consecutive elements in A_i , as can be observed in Figure 3.

We remark here that we use a hash function \mathcal{H} to generate keys for simplices because it is an efficient way to reuse keys (in case of multiple insertions and removals).

3.3 Size of the Critical Simplex Diagram

The number of nodes in each A_i is at most Ψ , and the total number of nodes in CSD, we denote by $|\text{CSD}|$, is at most $\Psi \cdot n$. Note that the number of edges in CSD is also at most $\Psi \cdot n$ since CSD is essentially a collection of star graphs.

Alternatively, we can bound the number of nodes by $|M|d$, where M as defined in Section 3.1 is the union of all M_h . The actual relation between Ψ and $|M|$ can be stated as follows:

$$\sum_{i=1}^n |\Psi_{\max}(i)| = \sum_{\sigma \in M} (d_\sigma + 1).$$

Further, in each node we store a filtration value (which requires $\log t$ bits) and a hashed value (which requires $\max_{h \in [t]} \log |M_h| \leq \log m$ bits). We can thus upper bound the space needed to store the nodes of CSD by $\Psi \cdot n(\log m + \log t)$ or by $|M| \cdot d(\log m + \log t)$. However, if we only store the filtration value and the hashed value at the center of the star graph then, we need only $|M|(d + \log m + \log t)$ space to store the nodes of CSD. In doing so, from any node of the star graph, we can still access/modify the filtration and hashed values in $\mathcal{O}(1)$ time. Thus, CSD matches the lower bound in Theorem 4, up to constant factors (as $m = \mathcal{O}(n^d)$).

In the case of CSD, we are interested in the value of Γ_0 and Ψ which we use to estimate the worst-case cost of basic operations in CSD, in the following subsection.

3.4 Operations on the Critical Simplex Diagram

Let us now analyze the cost of performing basic operations on CSD. First, we describe how to intersect arrays and update arrays in CSD as these are elementary operations on CSD which are required to perform basic operations on the simplicial complex that it represents. Next, we describe how to perform static queries such as the membership query. Following which we describe how to perform dynamic queries such as the insertion or removal of a simplex. Finally, we compare the efficiency of CSD with ST. We remark here that in order to perform the above operations efficiently, we will exploit the fact that the filtration value of a simplex that is not critical is equal to the minimum of the filtration values of its cofaces.

3.4.1 Elementary Operations to maintain the Critical Simplex Diagram

Below, we first discuss the implementation of the arrays in CSD using red-black trees, and thus we would have described how to search within an array and update an array of CSD. Next, we describe how to compute the intersection of the arrays in CSD, an operation needed to answer static queries.

Implementation of the arrays

We implement the arrays A_i using a variant of the red-black trees, and this means we can search, insert, and remove an element inside A_i in time $\mathcal{O}(\log |A_i|)$. Below we will discuss how to implement A_i^* and the same will hold for $A_i \setminus A_i^*$ which we treat separately. Each subarray of A_i^* which have the same filtration value i.e., the same first coordinate, is implemented using a red-black tree. Now these subarrays described above partition A_i^* and we can label each partition with the common first coordinate value of its elements. We represent the set of these partitions using a red-black tree by storing the partitions label. Therefore, each A_i is the union of two “doubly-composed” red-black trees. The way we search in A_i , is that we sequentially search in A_i^* , followed by $A_i \setminus A_i^*$.

Intersecting arrays

Let σ be a simplex of dimension d_σ and denote its vertices by $v_{\ell_0}, \dots, v_{\ell_{d_\sigma}}$. We will need to compute A_σ , defined as the intersection of $A_{\ell_0}, \dots, A_{\ell_{d_\sigma}}$, and A_σ^* , defined as the intersection of $A_{\ell_0}^*, \dots, A_{\ell_{d_\sigma}}^*$. To compute A_σ , we first find out the array with fewest elements amongst $A_{\ell_0}, \dots, A_{\ell_{d_\sigma}}$. Then, for each element x in that array, we search for x in the other d_σ arrays, which can be done in time $\mathcal{O}\left(d_\sigma \log \left(\max_i |A_{\ell_i}| \right)\right)$. Hence A_σ can be computed in time $\mathcal{O}\left(\left(\min_i |A_{\ell_i}| \right) d_\sigma \log \left(\max_i |A_{\ell_i}| \right)\right)$. As we have seen before, $|A_{\ell_i}| \leq \Psi \leq m$.

We can compute A_σ^* in the same way as described above for A_σ in time $\mathcal{O}\left(\left(\min_i |A_{\ell_i}^*| \right) d_\sigma \log \left(\max_i |A_{\ell_i}^*| \right)\right)$. As we have seen before, $|A_{\ell_i}^*| \leq \Gamma_0 \leq k$.

3.4.2 Static Operations on the Critical Simplex Diagram

The tree structure of ST provides an efficient representation to perform static operations. However, we show below that we are able to answer these static queries using CSD by only paying a multiplicative factor of Ψ (in the worst case) over the cost of performing the same operation in ST. In the case of the membership query, the multiplicative factor is reduced to Γ_0 .

Membership of a Simplex

We first observe that $\sigma \in K$ if and only if $A_\sigma^* \neq \emptyset$. This is because if $\sigma \in K$, then there exists a maximal simplex in K which contains σ . The star graph associated to this maximal simplex has nodes in all the $A_{\ell_i}^*$, and all those nodes have the same label. This implies that $A_\sigma^* \neq \emptyset$, and the converse is also true. It follows that determining if σ is in K reduces to computing

A_σ^* and checking whether it is non-empty. This procedure is very similar to the analogous procedure using SAL [BKT16]. Therefore, membership of a simplex can be determined in time $\mathcal{O}(d_\sigma \Gamma_0 \log k)$. Finally, we note that through the membership query, we are also able to decide if a simplex is maximal in the complex. We denote this new query by `is_maximal`, and will be used later for performing other operations.

Access Filtration Value

Given a simplex σ of K we want to access its filtration value $f(\sigma)$. We observe that $f(\sigma)$ is the minimal filtration value of the nodes in A_σ since the filtration function is monotone w.r.t. inclusion. Hence, accessing the filtration value of σ reduces to computing A_σ . Therefore, the filtration of a simplex can be determined in time $\mathcal{O}(d_\sigma \Psi \log \Psi)$.

For example, consider the CSD in Figure 3. We have to find the filtration value of $\sigma = [134]$ in the complex of Figure 1. We see that $A_1 \cap A_3 \cap A_4 = \{(3, 1), (5, 1)\}$. This means that the filtration value of the triangle is $f([134]) = \min(3, 5) = 3$. Finally, we note that through the filtration query, we are also able to decide if a simplex is critical in the complex. This new query, denoted by `is_critical`, will be used later for performing other operations.

Computing Filtration Value of Facets

Given a simplex $\sigma = v_{l_0}, \dots, v_{l_{d_\sigma}}$, we could procure the filtration value of its $d_\sigma + 1$ facets by the access filtration value query, and thus requiring a total running time of $\mathcal{O}(d_\sigma^2 \Psi \log \Psi)$. However, we will modify the access filtration value query to obtain filtration value of the $d_\sigma + 1$ facets of σ in running time of $\mathcal{O}(d_\sigma \Psi \log \Psi)$. Let $A_{\ell_r} = \underset{i}{\operatorname{argmin}} |A_{\ell_i}|$, for some $r \in \llbracket d_\sigma \rrbracket$. Let B be a subset of A_{ℓ_r} such that every element of A_{ℓ_r} which appears in exactly $d_\sigma - 1$ of the sets in $A_{\ell_0}, \dots, A_{\ell_{r-1}}, A_{\ell_{r+1}}, \dots, A_{\ell_{d_\sigma}}$ is in B . We can identify B in time $\mathcal{O}(|A_{\ell_r}| d_\sigma \log \Psi)$. To each entry (s, x) in B we associate the index of the set A_{ℓ_i} which does not contain (s, x) by a mapping g . We sort B based on g , and in case of ties based on the first coordinate. The filtration value of the facet $\sigma' = v_{\ell_0} \cdots v_{\ell_{j-1}} \cdot v_{\ell_{j+1}} \cdots v_{\ell_{d_\sigma}}$, is the minimal filtration value of the nodes in B which are mapped to j under g . If there are no nodes in B mapped to j under g then, the filtration value of σ' is $f(\sigma)$. This computation for all the $d_\sigma + 1$ facets requires a total time of $\mathcal{O}(|B| \log |B| + d_\sigma \log |B|) = \mathcal{O}((\Psi + d_\sigma) \log \Psi)$. Therefore the total running time is $\mathcal{O}(d_\sigma \Psi \log \Psi)$.

Computing Filtration Value of Cofaces of codimension 1

Given a simplex σ , we perform the access filtration value query to obtain all critical simplices that contain σ . We traverse the star graphs that contain the nodes in A_σ , to list these critical simplices in time $\mathcal{O}(d \Psi \log \Psi)$. From this list, we can compute the filtration value of all the cofaces of σ of codimension 1 in $\mathcal{O}(d \Psi \log \Psi)$ time.

3.4.3 Dynamic Operations on the Critical Simplex Diagram

Now, we will see how to perform dynamic operations on CSD. We note here that CSD is more suited to perform dynamic queries over ST because of its non-explicit representation, and this

means that the amount of information to be modified is always less than ST.

Lazy Insertion

Lazy insertion is the operation of inserting σ into the complex without checking if there are simplices in the complex which were previously critical but are now faces of σ with the same filtration value as $f(\sigma)$. Additionally, in the case of lazy insertion, we assume that the information about σ being a maximal simplex (or not) is known as part of the input. Lazy insertion will be extensively used in the later sections for preliminary construction of simplicial complexes. Lazy insertion in CSD requires $\mathcal{O}(d_\sigma \log \Psi)$ time (i.e., the cost of updating the arrays).

Insertion

Suppose we want to insert a simplex σ with filtration value $f(\sigma)$ such that any coface of σ in K has a filtration value larger than $f(\sigma)$. The insertion operation consists of first checking if σ is a maximal simplex in K by the `is_maximal` query. If σ is maximal, then we have to insert the simplex and remove or reallocate (based on filtration value) all simplices which were maximal simplices in K but are now faces of σ . If σ is not maximal, then we just have to lazy insert σ into the complex. We remark here that we do not need to remove the faces of σ which were previously critical simplices and had filtration value at least $f(\sigma)$ as their presence will not hinder any operation on CSD. Alternatively, we can think of performing a clean-up operation in parallel where such nodes are removed from CSD without affecting any other operation (this discussion is elaborated in the paragraph called ‘Robustness in Modification’ in Section 3.5).

Suppose σ is a maximal simplex then, insert the star graph corresponding to σ in $A_{\ell_0}^*, \dots, A_{\ell_{d_\sigma}}^*$. Updating the arrays A_{ℓ_i} takes time $\mathcal{O}(d_\sigma \log \Psi)$. Next, we have to check if there exist maximal simplices in K which are now faces of σ , and either remove them if their filtration value is equal to $f(\sigma)$ or move them outside $A_{\ell_i}^*$ if their filtration value is strictly less than that of $f(\sigma)$. We restrict our search for faces of σ which were previously critical by looking for every vertex v in σ , at the set of all maximal simplices which contain v , denoted by Z_v . We can compute Z_v in time $\mathcal{O}(d_\sigma \Gamma_0 \log \Gamma_0)$. Then, we compute $\bigcup_{v \in \sigma} Z_v$ whose size is at most $(d_\sigma + 1)\Gamma_0$ and check if any of these maximal simplices are faces in σ (can be done in $\mathcal{O}(d_\sigma^2 \Gamma_0)$ time). If such a face of σ in $\bigcup_{v \in \sigma} Z_v$ has filtration value equal to $f(\sigma)$ then, we remove that connected component. To remove all such connected components takes time $\mathcal{O}(d_\sigma^2 \Gamma_0 \log \Gamma_0)$. On the other hand, if filtration value of the face is less than $f(\sigma)$ then, we will have to move the node outside $A_{\ell_i}^*$ and place it appropriately to maintain the sorted structure of A_{ℓ_i} . To reallocate all such connected components takes time $\mathcal{O}(d_\sigma^2 \Gamma_0 \log \Psi)$. Summarizing, to handle removal of face or reallocating the concerned faces of σ which were previously maximal takes time at most $\mathcal{O}(d_\sigma^2 \Gamma_0 (\log \Gamma_0 + \log \Psi))$.

If σ is not a maximal simplex then, we insert the star graph corresponding to σ in $A_{\ell_0}, \dots, A_{\ell_{d_\sigma}}$. Updating the arrays A_{ℓ_i} takes time $\mathcal{O}(d_\sigma \log \Psi)$. Therefore, the total running time in this case is $\mathcal{O}(d_\sigma \log \Psi)$.

Therefore, the total time for insertion is $\mathcal{O}(d_\sigma^2 \Gamma_0 (\log \Gamma_0 + \log \Psi)) = \mathcal{O}(d_\sigma^2 \Gamma_0 \log \Psi)$.

Removal

To remove a face σ , we first perform an access filtration value query of σ (requires $\mathcal{O}(\Psi d_\sigma \log \Psi)$ time). We deal with the simplices in A_σ^* and $A_\sigma \setminus A_\sigma^*$ separately. For every simplex $\tau \in A_\sigma^*$, i.e., for every coface τ of σ in K which is a maximal simplex, we remove its corresponding star graph from the CSD. Since there are at most Γ_{d_σ} maximal simplices which contain σ , the above removal of star graphs can be done in $\mathcal{O}(\Gamma_{d_\sigma} d \log \Psi)$ time. Next, for each maximal simplex τ (containing σ) that we removed, and for every $i \in \llbracket d_\sigma \rrbracket$ we check if the facet of τ obtained by removing $v_{\ell_{i-1}}$ from τ , is a maximal simplex. If yes, we lazy insert the facet as a maximal simplex with the same filtration value as τ . If no, we lazy insert the facet as a non-maximal simplex with the same filtration value as τ . Note that in order to check if the above mentioned $d_\sigma + 1$ facets of τ are maximal, we do not have to make $d_\sigma + 1$ `is_maximal` queries, but can do the same checking in $\mathcal{O}(\Gamma_{d_\sigma} d \log \Psi)$ time by using the same idea that is described in the ‘computing filtration value of facets’ paragraph in Section 3.4.2. Additionally, we remark here that we can lazy insert the above selected facets of the the critical cofaces of σ , without checking if the facets themselves are critical because the argument made in the Insertion paragraph above for such cases apply here as well.

Next, for every simplex $\tau \in A_\sigma \setminus A_\sigma^*$ i.e., for every coface τ of σ in K which is a critical (not maximal) simplex, we replace its corresponding star graph by star graphs of its $d_\sigma + 1$ facets with the same filtration value, where the i^{th} facet is obtained by removing $v_{\ell_{i-1}}$ from τ . Introducing a star graph and updating the arrays A_{ℓ_i} takes time $\mathcal{O}(d_\tau \log \Psi)$. Further, if σ is a critical simplex (can be checked by `is_critical` query) then, we know that there is a connected component representing σ . We replace this star graph by the star graph for all its facets which have the same filtration value. Therefore, the total running time is $\mathcal{O}(dd_\sigma \Psi \log \Psi)$. We note here that as before, we lazy insert the above selected facets of the the critical cofaces of σ , without checking if the facets themselves are critical.

Therefore, the total time for insertion is $\mathcal{O}(\Psi dd_\sigma \log \Psi + \Gamma_{d_\sigma} d_\sigma d \log \Psi + \Gamma_{d_\sigma}^2 d \log \Psi) = \mathcal{O}((\Psi d_\sigma + \Gamma_{d_\sigma}^2) d \log \Psi)$.

Elementary Collapse

A simplex τ is collapsible through one of its faces σ , if τ is the only coface of σ . Such a pair (σ, τ) is called a free pair, and removing both faces of a free pair is an elementary collapse. Given a pair of simplices (σ, τ) , to check if it is a free pair is done by obtaining the list of all maximal simplices which contain σ , through the membership query (costs $\mathcal{O}(d_\sigma \Gamma_0 \log \Gamma_0)$ time) and then checking if τ is the only member in that list with codimension 1. If yes, then we remove τ from the CSD by just removing all the nodes in the corresponding arrays in time $\mathcal{O}(d_\tau \log \Psi)$. Next, for every facet σ' of τ other than σ , we check if σ' is a critical simplex (post removal of τ) by asking the `is_critical` query. If yes, we lazy insert σ' in time $\mathcal{O}(d_\sigma \log \Psi)$. Finally, if σ is a critical simplex then, we remove it in the same way we removed τ and for every facet of σ we similarly check if it is a critical simplex (post removal of σ) by asking the `is_critical` query. If yes, we lazy insert that facet in time $\mathcal{O}(d_\sigma \log \Psi)$. Thus, the total running time is $\mathcal{O}(d_\sigma(d_\sigma \Psi \log \Psi + \Gamma_0 \log \Gamma_0)) = \mathcal{O}(d_\sigma^2 \Psi \log \Psi)$.

3.4.4 Summary

We summarize in Table 1 the asymptotic cost of basic operations discussed above and compare it with ST, through which the efficiency of CSD is established.

	ST	CSD
Storage	$\Theta(m \log(nt))$	$\mathcal{O}(\Psi n \log(\Psi t))$
Membership of a simplex σ	$\Theta(d_\sigma \log n)$	$\mathcal{O}(d_\sigma \Gamma_0 \log k)$
Access Filtration Value	$\Theta(d_\sigma \log n)$	$\mathcal{O}(d_\sigma \Psi \log \Psi)$
Computing Filtration Value of Facets	$\mathcal{O}(d_\sigma^2 \log n)$	$\mathcal{O}(d_\sigma \Psi \log \Psi)$
Computing Filtration Value of Cofaces of codimension 1	$\Theta(nd_\sigma \log n)$	$\mathcal{O}(d \Psi \log \Psi)$
Lazy Insertion of a simplex σ	–	$\mathcal{O}(d_\sigma \log \Psi)$
Insertion of a simplex σ	$\mathcal{O}(2^{d_\sigma} d_\sigma \log n)$	$\mathcal{O}(d_\sigma^2 \Gamma_0 \log \Psi)$
Removal of a face	$\mathcal{O}(m \log n)$	$\mathcal{O}((\Psi d_\sigma + \Gamma_{d_\sigma}^2) d \log \Psi)$
Elementary Collapse	$\Theta(nd_\sigma \log n)$	$\mathcal{O}(d_\sigma^2 \Psi \log \Psi)$

Table 1: Cost of performing basic operations on CSD in comparison with ST.

If the number of critical simplices is not large then $|\text{CSD}|$ is smaller than $|\text{ST}|$. The number of critical simplices is small unless we associate unique filtration values to a significant fraction of the simplices. For this subsection, we will assume that the number of critical simplices is small (this assumption will be justified in the next subsection). In this case, we have Ψ to be small and thus the size of CSD is smaller than the size of ST.

We observe that while performing static queries, we pay a factor of Ψ or Γ_0 in the case of CSD over the cost of the same operation in ST. In the case of dynamic operations we observe that the dependence on the dimension is exponentially smaller in CSD than in ST. Therefore, even if the number of critical simplices is polynomial in the dimension then, there is an exponential gap between CSD and ST in both the storage and the efficiency of performing dynamic operations. Furthermore, in the case of insertion, CSD depends on Γ_0 and not Ψ (recall that $\Gamma_0 \leq \Psi$). Thus, the efficient insertion operation in CSD allows for fast construction of simplicial complexes, as we will see in future sections.

In short, CSD needs less storage than ST and performs dynamic operations more efficiently than ST while paying (mostly) a small multiplicative factor over ST in performing static queries. This is analogous to the trade-off between NFA (Non-deterministic Finite state Automaton) and DFA (Deterministic Finite state Automaton).

3.5 Performance of CSD

CSD has been designed to store filtrations of simplicial complexes but it can be used to store simplicial complexes without a filtration. In this case, $|M| = k$ and CSD requires $\mathcal{O}(kd \log k)$ memory space, which matches the lower bound in Theorem 2, when $k = n^{\mathcal{O}(1)}$. In this case, CSD is very similar to SAL [BKT16]. Marc Glisse and Sivaprasad S. [GS] have performed experiments on SAL and concluded that it is not only smaller in size but also faster than the Simplex Tree in performing insertion, removal, and edge contraction.

No	t	ST	CSD	Γ_0	Γ_0^{avg}	Ψ	Ψ^{avg}
0	0	10,508,486	179,521	115	17.9	115	17.9
1	10	10,508,486	490,071	115	17.9	329	49.0
2	25	10,508,486	618,003	115	17.9	429	61.8
3	100	10,508,486	728,245	115	17.9	723	72.8
4	500	10,508,486	765,583	115	17.9	839	76.5
5	2,000	10,508,486	774,496	115	17.9	860	77.4
6	10,000	10,508,486	777,373	115	17.9	865	77.7
7	25,000	10,508,486	778,151	115	17.9	865	77.8
8	100,000	10,508,486	778,319	115	17.9	866	77.8
9	1,000,000	10,508,486	778,343	115	17.9	866	77.8
10	10,000,000	10,508,486	778,343	115	17.9	866	77.8

Table 2: Values of $|CSD|$, Γ_0 , Ψ , and Ψ^{avg} for the simplicial complex generated from the above data set with increasing values of t . Additionally, we provide Γ_0^{avg} which is the number of maximal simplices that an average vertex contains and Ψ^{avg} which is the average size of A_i .

CSD is also a compact data structure to store filtrations, as its size matches (up to constant factors) the lower bound of $\Omega(|M|(d \log n + \log t))$ in Theorem 4. Moreover, if Ψ is small, CSD is not only a compact data structure since $|CSD|$ is upper bounded by Ψn , but, as shown in Table 1, CSD is also a very efficient data structure as all basic operations depend polynomially on d (as opposed to ST for which some operations depend exponentially on d).

As our analysis shows, we can express the complexity of CSD in terms of a parameter Ψ that reflects some “local complexity” of the simplicial complex. In the worst-case, $\Psi = \Omega(m)$ as it can be observed in the complete complex with each simplex having a unique filtration value. However we conjecture that, even if m is not small, Ψ remains small for a large class of simplicial complexes of practical interest. This conjecture is supported by the following experiment.

We considered a set of points obtained by sampling a Klein bottle in \mathbb{R}^5 and constructed its Rips filtration (see Section 4 for definition) using libraries provided by the GUDHI project [Pro15]. We computed Γ_0 and Ψ for various values of t . The resulting simplicial complex on 10,000 vertices is 17 dimensional and has 10,508,486 simplices of which 27,286 are maximal. We record in Table 2 below, the values of $|ST|$ and $|CSD|$ for the various filtration ranges of the Rips complex constructed above. In Figure 4 is a graphical illustration of the data.

We note from Table 2 that Γ_0 is significantly smaller than $k(\approx 2.7 \times 10^4)$, and also that Γ_0^{avg} is much smaller than Γ_0 . Also, from Figure 4, it is clear that there is an order of magnitude gap between $|CSD|$ and $|ST|$. Next, we note that Ψ is remarkably smaller than m (even notably smaller than n), and this implies efficient implementation of all operations. More importantly, we remark here that $\Psi^{\text{avg}} = |CSD|/n$ is at most 77.8 in the above experiment. Finally, we observe that despite increasing t at a rapid rate, $|CSD|$ grows very slowly after $t = 100$. This is because the set of all possible filtration values of the Rips complex is small. Therefore, even for small values of t the simplicial complex and its filtration is accurately captured by CSD.

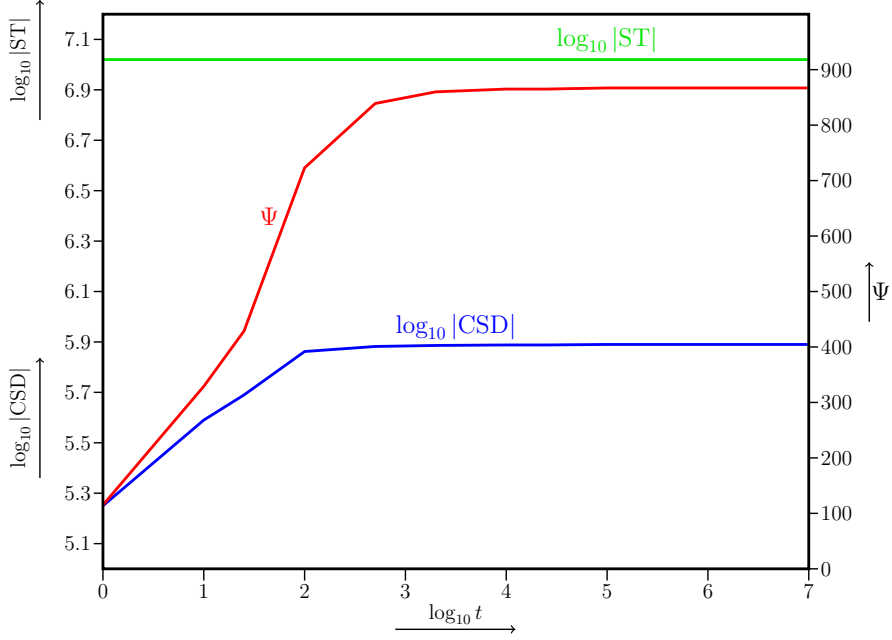


Figure 4: Values of $\log_{10} |ST|$, $\log_{10} |CSD|$, and Ψ for the simplicial complex generated from above data set with increasing values of t . The blue curve corresponds to $\log_{10} |CSD|$ on the left y axis plotted against $\log_{10} t$ on the x -axis. The red curve corresponds to Ψ on the right y axis plotted against $\log_{10} t$ on the x -axis. The green curve(line) corresponds to $\log_{10} |ST|$ on the left y axis plotted against $\log_{10} t$ on the x -axis.

Local Sensitivity of the Critical Simplex Diagram

It is worth noting that while the cost of basic operations are bounded using Γ_0 and Ψ , the actual cost is bounded by parameters such as $\min_i (|A_{\ell_i}^*|)$, $\min_i (|A_{\ell_i}|)$, and Z_v (introduced in the Insertion paragraph in Section 3.4.3) to get a better estimate on the cost of these operations. These parameters are indeed local. To begin with, $\min_i (|A_{\ell_i}^*|)$ captures the local information about a simplex σ sharing a vertex with other maximal simplices of the complex. More precisely, it is the minimum, over all the vertices of σ , of the largest number of maximal simplices that contain the vertex. If σ has a vertex which is contained in a few maximal simplices then, $\min_i (|A_{\ell_i}^*|)$ is small. Similarly, $\min_i (|A_{\ell_i}|)$ is the minimum, over all the vertices of σ , of the largest number of critical simplices that contain the vertex. This value depends not only on the structure of the filtration function but also on the filtration range. Finally, Z_v captures another local property of a simplex σ – the set of all maximal simplices that contain the vertex v . Therefore, CSD is sensitive to the local structure of the complex.

Robustness in Modification

We now demonstrate the robustness of CSD, i.e., its ability to perform queries *correctly* and *efficiently* even when it might have stored redundant data such as simplices which are not critical or multiple copies of the same simplex with different filtration values. Consider modifying the filtration value of some simplex $\sigma \in K$ from $f(\sigma)$ to $s_\sigma (< f(\sigma))$. In the case of ST, we will have to modify the filtration value inside the node containing σ and additionally check (and modify if needed) its faces in decreasing order of dimension. This requires time $\Theta(2^{d_\sigma} d_\sigma (\log n + \log t))$.

However, in the case of CSD, we can perform a lazy insertion of σ into CSD in time $\mathcal{O}(d_\sigma \log \Psi)$, and the data structure is robust to such an insertion. This is because, all the operations can be performed correctly and with the same[†] efficiency after the lazy insertion (even if some previously critical simplices need to be removed due to the lazy insertion of σ). For instance, consider the `is_critical` query on some simplex τ . If τ was a face of σ before modifying $f(\sigma)$ then, the *minimal* filtration value of the nodes in A_τ correctly gives the filtration value of τ as s_σ will now be one of the entries in A_τ . Otherwise, if τ was not a face of σ then the filtration value of τ remains unchanged, as the lazy insertion of σ has not introduced a new simplex, but only a new filtration value to an existing simplex. Therefore, we can think of using the data structure to manipulate simplicial complexes in very short time through a collection of lazy insertions and perform a clean-up operation at the end of the collection of lazy insertions, or even think of performing the clean-up operation in parallel to the lazy insertions. This idea has been applied in performing the insertion and removal operations as discussed in Section 3.4.3. We remark here that if we lazy insert r simplices then in the worst case, Ψ grows to $r + \Psi$. In other words, the presence of redundant simplices, implies that the efficiency dependence will now be on $r + \Psi$ instead of Ψ , but the redundancy will not affect the correctness of the operations.

3.6 A Sequence of Representations for Simplicial Complexes and their Filtrations

Boissonnat et al. [BKT16] in their paper on Simplex Array List described a sequence of data structures, each more powerful than the previous ones (but also bulkier). In that sequence of data structures $\langle \Lambda \rangle$, we had $\Lambda_i = i$ -SAL (SAL referred to earlier in this paper is equal to 1-SAL). Further, they note that in the i^{th} element of the sequence, every node which is not a leaf (sink) in the data structure corresponds to a unique i -simplex in the simplicial complex. Also for all i -SAL, $i \in \mathbb{N}$, they state that it is a NFA recognizing all the simplices in the complex. As one moves along the sequence, the size of the data structure blows up by a factor of d at each step. But in return, there is a gain in the efficiency of searching for simplices as the membership query depends on Γ_i which decreases as i increases.

We note here that CSD described in this paper is exactly the same as 0-SAL, when $t = 0$ (we ignore the structure of the connected component, which is a path in SAL but a star in CSD). Therefore, CSD supersedes 0-SAL. There is no change in representation of a simplex between SAL and CSD; instead we only store more simplices (i.e, all critical simplices) in CSD. Therefore, in the same vein as $\langle \Lambda \rangle$, we can define a sequence of data structures, each more powerful than the previous ones (but also bulkier). More formally, consider the sequence of data structures $\langle \Pi \rangle$, where $\Pi_0 = \text{CSD}$ and $\Pi_i = \bigcup_{j=0}^t i\text{-SAL}(M_j)$, for all $i \in \mathbb{N}$. Further, for all $i \in \mathbb{N}$, we will refer to the data structure Π_i by the name i -CSD (we will continue to refer to 0-CSD as CSD). As we move along the sequence Π , the size of the data structure blows up by a factor of d at each step. But in return, we gain efficiency in searching for simplices as the membership query depends on Γ_i which decreases as i increases. Additionally, we gain efficiency in accessing filtration value of a simplex as the complexity no longer depends on $\Psi = \Psi_0$ but on a smaller parameter, Ψ_i , which is the maximum number of critical cofaces that any i -simplex can have in the complex.

Marc Glisse and Sivaprasad S. implemented SAL [GS] for Data Set mentioned in Section 3.4, and then performed insertion and removal of random simplices, and contracted randomly

[†]up to constant additive factors in the worst case.

chosen edges. They observed that 1-SAL outperformed 0-SAL in low dimensions. However, 0-SAL performed better than 1-SAL in higher dimensions. Therefore, in similar vein, it would be worth exploring for which class of simplicial complexes, i -CSD is the best data structure in the CSD family (for every $i \in \mathbb{N}$).

4 Construction of Flag Complexes

The flag complex of an undirected graph G is defined as an abstract simplicial complex, whose simplices are the sets of vertices in the cliques of G . Let $(P, \|\cdot\|)$ be a metric space where P is a discrete point-set. Given a positive real number $r > 0$, the Rips complex is the abstract simplicial complex $\mathcal{R}^r(P)$ where a simplex $\sigma \in \mathcal{R}^r(P)$ if and only if $\|p - q\| \leq 2r$ for every pair of vertices of σ . Note that the Rips complex is a special case of a flag complex. Rips filtrations are widely used in Topological Data Analysis since they are easy to compute and they allow to robustly reconstruct the homology of a sample shape via the computation of its persistence diagram [CCG⁺09].

We will describe a specific filtration for Flag complexes which is of significant interest as it includes the Rips filtration. The filtration value of a vertex is 0. The filtration value of every edge in the complex is given as part of the input. The filtration value of a simplex of higher dimension is equal to the maximum of the filtration values of all the edges in the simplex.

4.1 Edge-Deletion Algorithm for Construction of Flag Complexes

Let G be the (weighted) graph of the simplicial complex K . Let Δ denote the maximum degree of the vertices of G . To represent K using ST, Boissonnat and Maria [BM14b] propose computing and inserting the ℓ -skeleton of K into the ST and incrementally increasing ℓ from 1 to d . Therefore, the time for construction of the ST representing the flag complex is $\mathcal{O}(mnd \log n)$.

To represent K using CSD, we propose an *edge-deletion* algorithm, which is significantly faster than the construction algorithm for ST. We recall that in Section 3.2 we defined S_h to denote the set of simplices in the complex with filtration value h .

Preprocessing Step. We first compute all maximal cliques in G in time $\mathcal{O}(k \cdot n^\omega)$ [MU04], where $\omega < 2.38$ [Gal14] is the matrix multiplication exponent, i.e., n^ω is the time needed to multiply two $n \times n$ matrices. We store these maximal simplices in a Prefix Tree (like MxST of [BKT16]). The filtration value given to the edges provides a natural ordering to the edges of the complex. We consider edges in descending order of their filtration value. Let e_i be the edge with the i^{th} highest filtration value. Recall that all simplices containing e_i are of filtration value $f(e_i)$ and are in $S_{f(e_i)}$. Fix $i = 1$.

Step 1. In this step, we would like to compute $M_{f(e_i)}$ in order to build CSD. A natural way to do that is by first computing $S_{f(e_i)}$, and then identifying the subset $M_{f(e_i)}$. Computing $S_{f(e_i)}$ requires time $\mathcal{O}(|S_{f(e_i)}| d \log n)$ and then computing $M_{f(e_i)}$ will require time $\tilde{\mathcal{O}}(|S_{f(e_i)}| \cdot d \cdot |M_{f(e_i)}|)$ using the best known algorithms in literature [Yel92, YJ93, Pri97, BP11]. However, we will not compute $M_{f(e_i)}$ from $S_{f(e_i)}$, but instead skip computing $S_{f(e_i)}$ and directly compute $M_{f(e_i)}$ to list all the maximal simplices only containing the edge e_i in

time $\mathcal{O}(|M_{f(e_i)}| \Delta^\omega)$ using the algorithm presented by Makino and Uno [MU04] on a subgraph of G in the following way. We build an induced subgraph H of G which contains the vertices of the edge e_i and all the vertices which are adjacent to *both* the vertices of e_i . We note that every maximal clique in H is a maximal clique in G containing the edge e_i , and vice versa. Therefore, if we run Makino and Uno's algorithm on H (which contains at most $\Delta + 1$ vertices), we obtain all the maximal cliques in G containing the edge e_i .

Step 2. Next, we recognize the maximal simplices of K in $M_{f(e_i)}$ in time $\mathcal{O}(|M_{f(e_i)}| d \log n)$ by checking each simplex σ in $M_{f(e_i)}$ with the Prefix tree built in the preprocessing step in time $\mathcal{O}(d_\sigma \log n)$ (per simplex). We remark here that all simplices in $M_{f(e_1)}$ are maximal simplices in K , since e_1 has the largest filtration value.

Step 3. We perform lazy insertion of simplices in $M_{f(e_i)}$ into the CSD and since we have identified the maximal simplices in $M_{f(e_i)}$, we know whether to insert them in A_j^* or not, within each A_j . This takes time $\mathcal{O}(|M_{f(e_i)}| d \log \Psi)$.

Step 4. Finally, we remove e_i from G , increment i by 1, and repeat the procedure from step 1 until G has no edges left.

This entire construction takes time $\mathcal{O}(|M|(\Delta^\omega + d \log(kd\Psi))) = \mathcal{O}(|M|n^{2.38})$, which is significantly better than that of constructing a representation of K by ST (which required time $\mathcal{O}(mnd \log n)$), as $|M|$ can be considerably (exponentially) smaller than m .

4.2 Performance of CSD for Flag Complexes

We would like to note here that the case when $k = \mathcal{O}(n)$, was argued to be of particular interest by Boissonnat et al. [BKT16]. It can be observed in flag complexes, constructed from planar graphs and expanders [ELS10], and in general, from nowhere dense graphs [GKS13], and also from chordal graphs [Gol80]. Generalizing, they noted that for all flag complexes constructed from graphs with degeneracy $\mathcal{O}(\log n)$ (degeneracy is the smallest integer r such that every subgraph has a vertex of degree at most r), we have that $k = n^{\mathcal{O}(1)}$ [ELS10]. We add to this list of observations by noting that the flag complexes of K_ℓ -free graphs have at most $\max\{n, n\Delta^{\ell-2}/2^{\ell-2}\}$ maximal simplices [Pri95], where Δ is the maximum degree of any vertex in the graph. Thus, when Δ and ℓ are constants, we have $k = \mathcal{O}(n)$. Finally, we note that the flag complexes of Helly circular-arc (respectively, circle) graphs [Gav74, Dur03], and boxicity-2 graphs [Spi03] have $k = n^{\mathcal{O}(1)}$ from Corollary 4 of [RS07]. This encompasses a large class of complexes encountered in practice and if the number of maximal simplices is small, CSD is a very efficient data structure as $\Gamma_0 \leq k$.

4.3 Adaptation to Simplicial Maps

A map $F : K \rightarrow K'$ is simplicial if for every simplex $\sigma = \{v_0, v_1, \dots, v_k\}$ in K , $F(\sigma) = \{F(v_0), F(v_1), \dots, F(v_k)\}$ is a simplex in K' . In this subsection, we discuss the adaptation of CSD to handle simplicial maps. For the purpose of demonstration, we consider Dey et al.'s [DFW14] application of simplicial maps to topological data analysis over Rips complexes. They construct a sequence of Rips simplicial complexes, $\langle K_i^{\alpha_i} \rangle$ connected by simplicial maps, where

α_i is the Rips parameter. The vertex set of $K_{i+1}^{\alpha_{i+1}}$ is obtained by extracting a subset (net) of the vertex set of $K_i^{\alpha_i}$. They define a map π through which they map each vertex of $K_i^{\alpha_i}$ to its closest vertex in $K_{i+1}^{\alpha_{i+1}}$. Next, they take $\alpha_{i+1} > \alpha_i$ so that the image of the edges of $K_i^{\alpha_i}$ are edges in $K_{i+1}^{\alpha_{i+1}}$. Since the complexes are Rips complexes, the image of all the simplices of $K_i^{\alpha_i}$ are in $K_{i+1}^{\alpha_{i+1}}$. To implement this procedure using CSD, we need to collapse the vertices of $\text{vert}(K_i^{\alpha_i}) \setminus \text{vert}(K_{i+1}^{\alpha_{i+1}})$ onto the vertices of $K_{i+1}^{\alpha_{i+1}}$ as given by π . First, we remark that this procedure is very expensive using ST, as vertex collapse is more expensive than the edge contraction operation, and Boissonnat and Maria [BM14b] provide an essentially optimal algorithm running in $\mathcal{O}(md \log n)$ time for edge contraction.

In the case of CSD, we will assume that $(|\text{vert}(K_i^{\alpha_i})| - |\text{vert}(K_{i+1}^{\alpha_{i+1}})|)$ is small, as otherwise, we could reconstruct $K_{i+1}^{\alpha_{i+1}}$ entirely from scratch using the fast edge-deletion algorithm. For every vertex v in $\text{vert}(K_i^{\alpha_i}) \setminus \text{vert}(K_{i+1}^{\alpha_{i+1}})$, we first build a set T_v from the set of maximal simplices containing v as follows. If σ is a maximal simplex containing v , then we include the simplex $\tau = \sigma \cap \text{vert}(K_{i+1}^{\alpha_{i+1}})$ in T_v . Let $T = \cup T_v$. Next, for every vertex v in $\text{vert}(K_i^{\alpha_i}) \setminus \text{vert}(K_{i+1}^{\alpha_{i+1}})$, we remove all the nodes in A_v and its neighbors in other arrays. For every simplex in T , we perform the `is_critical` query. If the simplex was not critical in $K_i^{\alpha_i}$ but is critical in $K_{i+1}^{\alpha_{i+1}}$, then we lazy insert the simplex. On the other hand, if the simplex was critical in $K_i^{\alpha_i}$, then we check if it is maximal in $K_{i+1}^{\alpha_{i+1}}$ and reallocate appropriately. The total cost of performing the above procedure is $\mathcal{O}(|T|d\Psi \log \Psi)$, where $|T| = \mathcal{O}(\Gamma_0(|\text{vert}(K_i^{\alpha_i})| - |\text{vert}(K_{i+1}^{\alpha_{i+1}})|))$. This running time is significantly better than that of implementing the simplicial map using ST, as Ψ , and consequently Γ_0 , may be considerably (exponentially) smaller than m . Adding to the above argument the benefit of the considerably (exponentially) smaller size of CSD, it is clear that CSD better supports the implementation of simplicial maps than ST.

5 Construction of Relaxed Delaunay Complexes

Let Q be a finite subset of a metric space $(P, \|\cdot\|)$ where P is a discrete point-set. Given a relaxation parameter $\rho \geq 0$, we define the notion of being ‘witnessed’ as follows. A simplex $\sigma = \{q_0, \dots, q_{d_\sigma}\} \subseteq Q$ belongs to the relaxed Delaunay complex[‡] $\text{Del}^\rho(Q, P)$ [dS08, BDG15] if and only if there exists $x \in P$ such that for all $q_i \in \sigma$, and for all $q \in Q$ the following holds:

$$\|x - q_i\| \leq \|x - q\| + \rho$$

The parameter ρ defines a filtration on the relaxed Delaunay complexes, which has been used in topological data analysis. More explicitly, the filtration value of a simplex σ in $\text{Del}^\rho(Q, P)$ is the smallest $\rho' \leq \rho$, such that σ is in $\text{Del}^{\rho'}(Q, P)$. For this entire section, we assume that the filtration range is $\llbracket t \rrbracket$.

We define a matrix D of size $|P| \times |Q|$ as follows. For every $x \in P$ and $\ell \in \llbracket |Q| \rrbracket$ let $D(x, \ell)$ denote the ℓ^{th} nearest neighbor of x in Q (ties are broken arbitrarily). For every $x \in P$, $i \in \llbracket t \rrbracket$, let ℓ_x^i be the largest integer such that $|\|x - D(x, 1)\| - \|x - D(x, \ell_x^i)\|| \leq \rho i/t$. Let $\sigma_x^i = \{D(x, 1), D(x, 2), \dots, D(x, \ell_x^i)\}$ and let $W = \{\sigma_x^i | x \in P, i \in \llbracket t \rrbracket\}$. We note below that M (recall notation from Section 3.1), the set of critical simplices in the complex is contained in W .

Lemma 5. $M \subseteq W$.

[‡]This complex was referred to as the ρ -relaxed strong Delaunay complex in [dS08] and as the ρ -relaxed Delaunay complex in [BDG15].

Proof. Let $\tau = \{v_0, \dots, v_{d_\tau}\}$ be a critical simplex in M with filtration value $f(\tau)$. By definition of $\text{Del}^\rho(Q, P)$, we have that there exists a point $x \in P$ which $(f(\tau)\rho/t)$ -strongly witnesses τ . Since τ is critical, we have that for every $q \in Q \setminus \tau$, the following holds.

$$\forall i \in \llbracket d_\tau \rrbracket, \quad \left| \|x - v_i\| - \|x - q\| \right| > \rho f(\tau)/t.$$

Therefore, we have that for every $i \in [d_\tau + 1]$, $A(x, \ell_x^i) \in \tau$, or more precisely, $\sigma_x^{d_\tau+1} = \tau$. \square

The above lemma provides a characterization of $\text{Del}^\rho(Q, P)$: it can have at most $|P|(d+1)$ critical simplices. We note here that typically P is a relatively small set. For example, in the experiments performed by Boissonnat and Maria (Table 1 of [BM14b]), we note that the cardinality of the witness set is about a few ten thousands while the number of simplices in the complex is over a hundred million. Therefore, this provides practical evidence of the compact representation of $\text{Del}^\rho(Q, P)$ through CSD.

Under the assumption that for any x, ℓ , $D(x, \ell)$ could be computed in $\mathcal{O}(1)$ time (i.e., D is computed as part of the preprocessing), Boissonnat and Maria [BM14b] described an algorithm to construct the ST representation of the relaxed witness complex. Their algorithm can be easily adapted to construct $\text{Del}_w^\rho(Q, P)$ in time $\mathcal{O}(tmd \log n)$.

In the case of CSD, we propose a new *matrix-parsing* algorithm which builds $\text{Del}^\rho(Q, P)$ in time $\mathcal{O}(|P|d^2 \log \Psi)$ (assuming an oracle to access D). It is easy to see that all the simplices in W can be constructed in $\mathcal{O}(|W|d \log n) = \mathcal{O}(|P|d^2 \log n)$ time by sequentially computing the simplices σ_x^i for all the $x \in P$, i.e., by parsing the matrix D one row at a time. From the discussions about the robustness of CSD discussed in Section 3.5, we know that we could lazy insert all the simplices in W to the CSD and it would behave exactly like in the scenario wherein only the simplices in M (which is a subset of W) are inserted. This lazy insertion of all the simplices in W can be done in time $\mathcal{O}(|P|d^2 \log \Psi)$. After the construction, we may perform a clean-up operation to remove the redundant simplices that were inserted.

6 Conclusion

In this paper, we introduce a new data structure called the Critical Simplex Diagram (CSD) to represent filtrations of simplicial complexes. In this data structure, we store only those simplices which are critical with respect to the filtration value, i.e., we store a simplex if and only if all its cofaces are of a (strictly) higher filtration value than the filtration value of the simplex itself. We then show how to efficiently perform basic operations on simplicial complexes by only storing these (critical) simplices. This is summarized in Table 1. Finally, we showed how to (quickly) construct the CSD representation of flag complexes and relaxed Delaunay complexes.

As a future direction of research, we would like to obtain better bounds on Ψ and Γ_i for specific complexes such as the Rips complex or the relaxed Delaunay complex by assuming some notion of geometric regularity. Also, it would be interesting to obtain lower bounds on the various query times (such as membership, insertion/removal), by assuming an optimal storage of $\mathcal{O}(\kappa d \log n)$ ($\kappa = |M|$ is the number of critical simplices). From the standpoint of practice, we would like to find fast construction algorithms under the CSD representation for other simplicial complexes of interest such as the alpha complex and the relaxed witness complex. Finally, we would like to implement this data structure and check its performance versus the Simplex Tree in practice.

Acknowledgements

We would like to thank the anonymous reviewers whose comments helped us improve the presentation of the paper.

References

- [ABD⁺12] Javier Arsuaga, Nils A. Baas, Daniel DeWoskin, Hideaki Mizuno, Aleksandr Pankov, and Catherine Park. Topological analysis of gene expression arrays identifies high risk molecular subtypes in breast cancer. *Appl. Algebra Eng. Commun. Comput.*, 23(1-2):3–15, 2012.
- [ALS12] Dominique Attali, André Lieutier, and David Salinas. Efficient data structure for representing and simplifying simplicial complexes in high dimensions. *Int. J. Comput. Geometry Appl.*, 22(4):279–304, 2012.
- [BDG15] Jean-Daniel Boissonnat, Ramsay Dyer, and Arijit Ghosh. A probabilistic approach to reducing algebraic complexity of delaunay triangulations. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 595–606, 2015.
- [BDM15] Jean-Daniel Boissonnat, Tamal K. Dey, and Clément Maria. The compressed annotation matrix: An efficient data structure for computing persistent cohomology. *Algorithmica*, 73(3):607–619, 2015.
- [BM14a] Jean-Daniel Boissonnat and Clément Maria. Computing persistent homology with various coefficient fields in a single pass. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 185–196, 2014.
- [BM14b] Jean-Daniel Boissonnat and Clément Maria. The simplex tree: An efficient data structure for general simplicial complexes. *Algorithmica*, 70(3):406–427, 2014.
- [BP11] Roberto J. Bayardo and Biswanath Panda. Fast algorithms for finding extremal sets. In *Proceedings of the Eleventh SIAM International Conference on Data Mining, SDM 2011, April 28-30, 2011, Mesa, Arizona, USA*, pages 25–34, 2011.
- [BKT16] Jean-Daniel Boissonnat, Karthik C. S., and Sébastien Tavenas. Building efficient and compact data structures for simplicial complexes. *Algorithmica*, pages 1–38, 2016. <http://dx.doi.org/10.1007/s00453-016-0207-y>.
- [CCG⁺09] Frédéric Chazal, David Cohen-Steiner, Marc Glisse, Leonidas J. Guibas, and Steve Oudot. Proximity of persistence modules and their diagrams. In *Proceedings of the 25th ACM Symposium on Computational Geometry, Aarhus, Denmark, June 8-10, 2009*, pages 237–246, 2009.
- [CCR13] Joseph Minhow Chan, Gunnar Carlsson, and Raul Rabadan. Topology of viral evolution. *Proceedings of the National Academy of Sciences*, 110(46):18566–18571, 2013.

- [CIIdSZ08] Gunnar E. Carlsson, Tigran Ishkhanov, Vin de Silva, and Afra Zomorodian. On the local behavior of spaces of natural images. *International Journal of Computer Vision*, 76(1):1–12, 2008.
- [DFW14] Tamal K. Dey, Fengtao Fan, and Yusu Wang. Computing topological persistence for simplicial maps. In *30th Annual Symposium on Computational Geometry, SOCG’14, Kyoto, Japan, June 08 - 11, 2014*, page 345, 2014.
- [dS08] Vin de Silva. A weak characterisation of the delaunay triangulation. *Geometriae Dedicata*, 135(1):39–64, 2008.
- [dSG07] V. de Silva and R. Ghrist. Coverage in sensor networks via persistent homology. In *Algebraic & Geometric Topology* 7, pages 339–358, 2007.
- [Dur03] Guillermo Durn. Some new results on circle graphs. *Matemtica Contemporanea*, 2003.
- [EH10] Herbert Edelsbrunner and John Harer. *Computational Topology - an Introduction*. American Mathematical Society, 2010.
- [ELS10] David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *Algorithms and Computation - 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I*, pages 403–414, 2010.
- [Gal14] François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC ’14, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014.
- [Gav74] Fanica Gavril. Algorithms on circular-arc graphs. *Networks*, 4(4):357–369, 1974.
- [GKS13] Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Characterisations of nowhere dense graphs (invited talk). In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12-14, 2013, Guwahati, India*, pages 21–40, 2013.
- [Gol80] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Computer science and applied mathematics. Academic Press, New York, 1980.
- [GS] M. Glisse and S. Sivaprasad. Private communication.
- [MU04] Kazuhisa Makino and Takeaki Uno. New algorithms for enumerating all maximal cliques. In *Algorithm Theory - SWAT 2004, 9th Scandinavian Workshop on Algorithm Theory, Humlebaek, Denmark, July 8-10, 2004, Proceedings*, pages 260–272, 2004.
- [PC14] Jose A. Perea and Gunnar Carlsson. A klein-bottle-based dictionary for texture representation. *International Journal of Computer Vision*, 107(1):75–97, 2014.
- [Pri95] Erich Prisner. Graphs with few cliques. In *7th Quadrennial International Conference on the Theory and Applications of Graphs, Graph Theory, Combinatorics, and Applications*, pages 945–956, 1995.
- [Pri97] Paul Pritchard. An old sub-quadratic algorithm for finding extremal sets. *Inf. Process. Lett.*, 62(6):329–334, 1997.

- [Pro15] The GUDHI Project. *GUDHI User and Reference Manual*. GUDHI Editorial Board, 2015.
- [RS07] Bill Rosgen and Lorna Stewart. Complexity results on graphs with few cliques. *Discrete Mathematics & Theoretical Computer Science*, 9(1), 2007.
- [Spi03] J.P. Spinrad. *Efficient Graph Representations.: The Fields Institute for Research in Mathematical Sciences*. Fields Institute monographs. American Mathematical Soc., 2003.
- [Yel92] Daniel M. Yellin. Algorithms for subset testing and finding maximal sets. In *Proceedings of the Third Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 27-29 January 1992, Orlando, Florida.*, pages 386–392, 1992.
- [YJ93] Daniel M. Yellin and Charanjit S. Jutla. Finding extremal sets in less than quadratic time. *Inf. Process. Lett.*, 48(1):29–34, 1993.